A Quick Overview of Python

Week 1

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Introduction to Python Programming

What is Python?

- Python is a high-level, interpreted general-purpose multi-paradigm programming language.

Introduction to Python Programming

What is Python?

- Python is a high-level, interpreted general-purpose multi-paradigm programming language.

Why learn Python?

- Language for web development, data analysis, machine learning, and scripting.

Introduction to Python Programming

What is Python?

- Python is a high-level, interpreted general-purpose multi-paradigm programming language.

Why learn Python?

- Language for web development, data analysis, machine learning, and scripting.
- User-friendly syntax which can quickly write programs and easily interface with high-performance libraries
- Provides rich library support for many applications

Introduction to Python Programming

What is Python?

- Python is a high-level, interpreted general-purpose multi-paradigm programming language.

Why learn Python?

- Language for web development, data analysis, machine learning, and scripting.
- User-friendly syntax which can quickly write programs and easily interface with high-performance libraries
- Provides rich library support for many applications
- A popular and extensively used language

Python

- This short introduction does not aim to cover every detailed aspect of Python, but rather the basic Python syntax/features in order to develop algorithms to fulfil the assignment tasks in this course.

Python

- This short introduction does not aim to cover every detailed aspect of Python, but rather the basic Python syntax/features in order to develop algorithms to fulfil the assignment tasks in this course.
- You are encouraged to learn and practice more advanced Python syntax/features.
    - `https://docs.python.org/3/tutorial/`
    - `https://www.w3schools.com/python/`
    - `https://cgi.cse.unsw.edu.au/~cs2041/25T1/topic/python_intro/slides`
    - Google search ‘Python programming‘ or ‘Introduction to Python programming‘

Write Your First Python Program

```python
print("Welcome to software security analysis course!")
```

A Hello World example under Software-Security-Analysis:

`https://github.com/SVF-tools/Software-Security-Analysis/blob/main/HelloWorld/hello.py`

If Statements in Python

```
x = int(input("Please enter an integer: "))
Please enter an integer: 42
if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print("Single")
else:
    print('More')
```

An if statement example from the Python docs:

`https://docs.python.org/3/tutorial/controlflow.html#if-statements`

For Loops in Python

```
words = ['cat', 'window', 'defenestrate']
for i in range(len(words)):
    print(words[i], len(words[i]))
```

For Loops in Python

```python
words = ['cat', 'window', 'defenestrate']
for i in range(len(words)):
    print(words[i], len(words[i]))
for w in words:
    print(w, len(w))
```

A for loop example from the Python docs:

`https://docs.python.org/3/tutorial/controlflow.html#for-statementss`

Containers/Collections

```
#Python lists
node_ids = []
node_ids.append(1)
node_ids.append(2)
node_ids.append(2)
for i in node_ids:
    print(i)
```

Containers/Collections

```
#Python lists
node_ids = []
node_ids.append(1)
node_ids.append(2)
node_ids.append(2)
for i in node_ids:
    print(i)
```

```
#Python sets
node_ids = set()
node_ids.add(1)
node_ids.add(2)
node_ids.add(2)
for i in node_ids:
    print(i)
```

Functions in Python

```
def fib(n):    # write Fibonacci series less than n
    """Return a Fibonacci series less than n."""
    series = []
    a, b = 0, 1
    while a < n:
        series.append(a)
        a, b = b, a+b
print(fib(2000))
```

Functions in Python

```python
def fib(n):    # write Fibonacci series less than n
    """Return a Fibonacci series less than n."""
    series = []
    a, b = 0, 1
    while a < n:
        series.append(a)
        a, b = b, a+b
print(fib(2000))
# An alternative function definition with the typing library
from typing import List
def fib(n: int) -> List[int]:
    ...
```

A function example from the Python docs:

`https://docs.python.org/3/tutorial/controlflow.html#defining-functions`

Python Classes and Objects

- Python objects: everything in Python is an object, there are no primitive types.
- A Python class is a template for objects, and an object is an instance of a class.
- All methods are public by default, a _ prefixed in the function name is used for protected methods or __ for private methods.

Python Classes and Objects

- Python objects: everything in Python is an object, there are no primitive types.
- A Python class is a template for objects, and an object is an instance of a class.
- All methods are public by default, a _ prefixed in the function name is used for protected methods or __ for private methods.

```
class Graph:
    def __init__(self, n: int, e: int):
        self.num_of_nodes: int = n
        self.num_of_edges: int = e
    def get_num_of_nodes(self) -> int:
        return self.num_of_nodes
    def set_num_of_nodes(self, n: int):
        return self.nodes
    def get_paths(self) -> Set[str]:
        return self.paths
```

Python Classes and Objects

- Python objects: everything in Python is an object, there are no primitive types.
- A Python class is a template for objects, and an object is an instance of a class.
- All methods are public by default, a _ prefixed in the function name is used for protected methods or __ for private methods.

```python
class Graph:
    def __init__(self, n: int, e: int):
        self.num_of_nodes: int = n
        self.num_of_edges: int = e
    def get_num_of_nodes(self) -> int:
        return self.num_of_nodes
    def set_num_of_nodes(self, n: int):
        return self.nodes
    def get_paths(self) -> Set[str]:
        return self.paths
```

```python
graph_obj = Graph(5, 10)
print(graph_obj.get_num_of_nodes)
```

Building a Graph with more Functionality

```python
class Node:
    def __init__(self, i: int):
        self.node_id = i
        self.out_edges = set()
    def get_node_id(self) -> int:
        return self.node_id
    def get_out_edges(self) -> Set[Edge]:
        return self.out_edges
class Edge:
    def __init__(self, s: Node, d: Node):
        self.src = s
        self.dst = d
    def get_src(self) -> Node:
        return self.src
    def get_dst(self) -> Node:
        return self.dst
```

Building a Graph with more Functionality

```python
class Node:
    def __init__(self, i: int):
        self.node_id = i
        self.out_edges = set()
    def get_node_id(self) -> int:
        return self.node_id
    def get_out_edges(self) -> Set[Edge]:
        return self.out_edges
class Edge:
    def __init__(self, s: Node, d: Node):
        self.src = s
        self.dst = d
    def get_src(self) -> Node:
        return self.src
    def get_dst(self) -> Node:
        return self.dst
```

```python
class Graph:
    def __init__(self):
        self.nodes: Set[Node] = set()
    def get_nodes(self) -> Set[Node]:
        return self.nodes
src = Node(1)
dst = Node(2)
edge = Edge(src, dst)
# add src's outgoing edge
src.get_out_edges().add(edge)
# create a graph object
graph = Graph()
# add two nodes into the graph
graph.get_nodes().add(src)
graph.get_nodes().add(dst)
```

Debugging Your Python Programs

- VSCode (`https://code.visualstudio.com/docs/python/debugging`)
- PDB (`https://docs.python.org/3/library/pdb.html`)
- Other tactics, such as printing your results (`https://adamj.eu/tech/2021/10/08/tips-for-debugging-with-print/`)